In [2]:
```
%matplotlib notebook
import numpy as np
import sympy as sym
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sci
from scipy.optimize import curve_fit
from scipy.constants import Boltzmann
from scipy.constants import e
from matplotlib import rc
from IPython.display import display, Math, Markdown
rc('font', **{'family': 'serif', 'serif': ['Computer Modern'], 'size': 14})
rc('text', usetex=True)
path = "C:/Users/Thomas/Laboratorio_Intermedio/Torsional Oscillator/Data/"
path_ = "C:/Users/Thomas/Laboratorio_Intermedio/Torsional Oscillator/Curves/"
```

Bitácora - Experimento del Oscilador Torsional

Hecho por: Simón Felipe Jimenez Botero & Thomas Andrade Hernández.

En el presente documento se encuentran desglozados los resultados de las mediciones asociadas al montaje experimental del Oscilador Torsional. A lo largo de este código se podrán observar los resultados de cada medición, gráficas e información relacionada a cada una de las regresiones que se realicen, sumado a una breve explicación previa acerca de aquello que estamos haciendo, cómo lo estamos haciendo y alguna que otra cosa a tener presente para su replicación.

In [3]:
```
def linear(X, a, b):
    return a*X + b
```

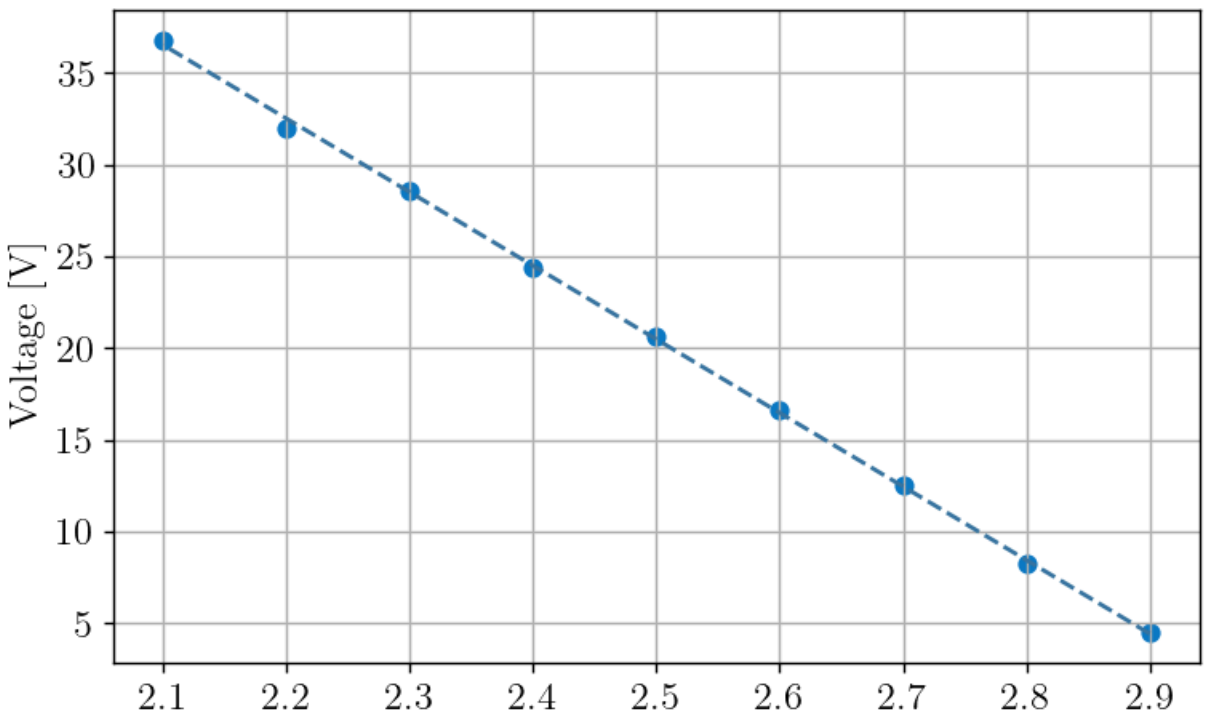
Primera Actividad: Calibración del Rotor.

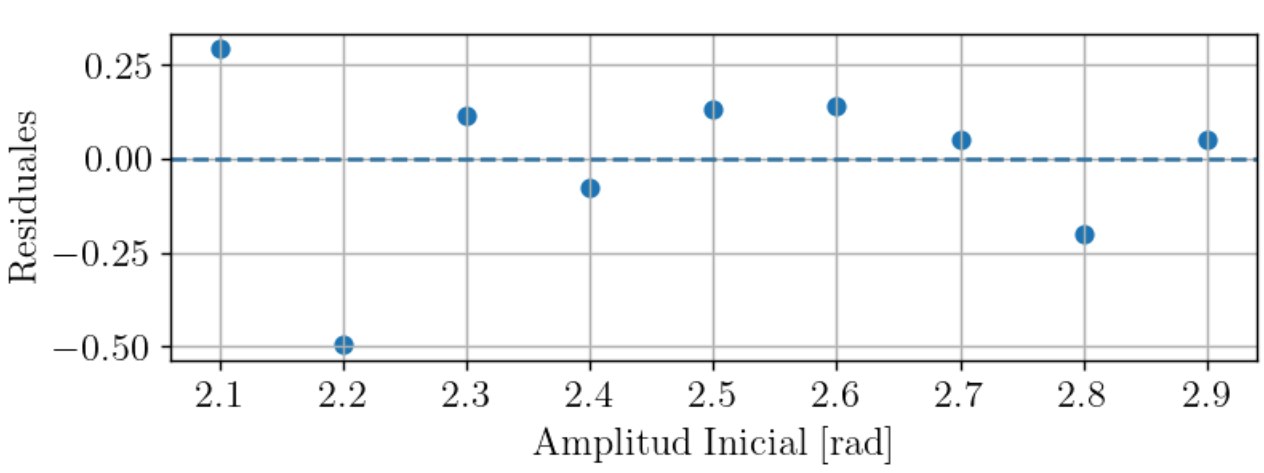
Para poder llevar a cabo todos los cálculos del experimento, es necesario establecer un valor para el ángulo inicial θ el cual genere una señal de voltaje pico a pico V_{pp} en el que la magnitud sea nula. Para esto, se registran los datos del voltaje pico a pico en función de la amplitud del ángulo inicial:

In [4]:
```
rotor_calib = "Rotor_Calibration.xlsx"
rotor_data = pd.read_excel(path + rotor_calib)
```

In [5]:
```
angle = rotor_data["Angle [rad]"].to_numpy()
allangle = np.linspace(angle[0], angle[-1], 100)
voltage = rotor_data["Vpp [mV]"].to_numpy()*1/100

voltage_coeff, voltage_cov = curve_fit(linear, angle, voltage)
voltage_unc = np.sqrt(np.diag(voltage_cov))
voltage_residues = voltage - linear(angle, *voltage_coeff)
```

In [6]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(angle, voltage, color = "#0C79C4")
axis[0].plot(allangle, linear(allangle, *voltage_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Voltage [V]")
axis[0].grid(True)


axis[1].scatter(angle, voltage_residues)
axis[1].set_ylabel("Residuales")
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_xlabel("Amplitud Inicial [rad]")
axis[1].grid(True)
```





In [7]:
```
angle_zero = -voltage_coeff[1]/voltage_coeff[0]
display(Markdown(r"El valor del ángulo asociado al cero de voltage para el rotor es de ${}$ radianes.".format(angle_zer
```

El valor del ángulo asociado al cero de voltage para el rotor es de 3.010530983131374 radianes.

Ya con este valor en mente, todas las actividades que vienen a continuación se deben llevar a cabo para esta configuración inicial.

> **Segunda Actividad: Aplicación del Torque Mecánico.**

Para llevar a cabo esta actividad, se hizo uso del sistema de poleas que está integrado en los laterales del montaje. Una vez conectado el sistema de manera adecuada **(considerando el sentido en el que se dirigen los torques de cada polea para evitar que se cancele el uno con el otro)**. Los pasos a seguir son:

1) Tras haber conectado de manera adecuada cada masa al sistema, medir la nueva posición de equilibrio para el sistema una vez se establecen las bases para las masas. Posterior a esto, determine el **radio del eje del rotor (?)** sobre el cual se encuentra enganchado el hilo. El valor en cuestión debe estar entre 12.7 mm y 25.4 mm. ¿Por qué se oscila entre estos valores?

- **Comentario:**

2) Comience por colocar ahora, en las bases laterales, masas de 500 gramos en cada una. Posteriormente, **incremente de manera coherente (?)** hasta llegar a 900 gramos, anotando en cada caso el nuevo punto de equilibrio.

- **Comentario:**

3) Repita este procedimiento para el caso en el que se invierte el sentido en que se amarraron los hilos de las poleas, para así obtener un barrido en las zonas negativas.

- **Comentario:**

4) Graficar el toque generado por las masas τ con respecto al cambio en el ángulo de equilibrio $\Delta\theta$. Partiendo de esto, determinar la constante de torsión κ, teniendo en cuenta las siguientes consideraciones teóricas:

Notar que el torque generado por las masas en las poleas, si es aplicado de forma correcta, tiene como magnitud:

$$\tau = 2rmg$$

donde r es el radio de las poleas y el 2 surge de la suma de los torques de ambas poleas, las cuales tendrán siempre los mismos valores de masa pendiendo de las mismas. También, el torque general del sistema se puede obtener como:

$$\tau = -\kappa\theta$$

de forma análoga con el sistema masa resorte. Así, igualando ambas expresiones es posible determinar que:

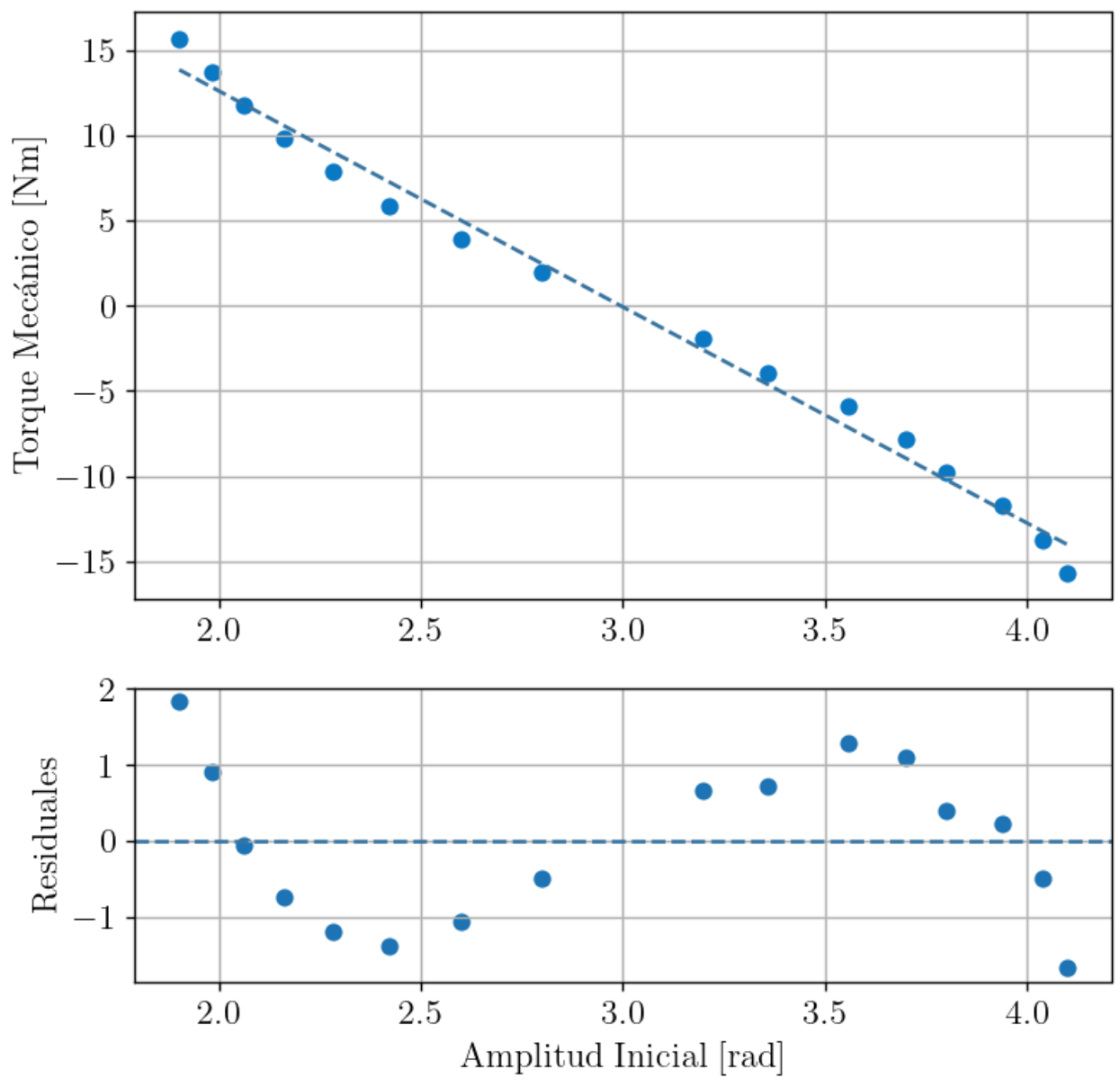
$$2rmg = -\kappa\theta$$

que a nivel práctico es realizar una gráfica de la forma de las masas en cada soporte m contra el cambio en el ángulo de equilibrio $\Delta\theta$. El cálculo del valor es el siguiente:

In [8]:
```
mechanic_torq = "Mechanic_Torque.xlsx"
mechanic_data = pd.read_excel(path + mechanic_torq)
```

In [9]:
```
radius = 1. # m
g = 9.81 # m/s^2
plus_torque = mechanic_data["Mass (+) [g]"].to_numpy()*(2*g*radius/1000)
plus_angle = mechanic_data["Angle (+) [rad]"].to_numpy()
minus_torque = mechanic_data["Mass (-) [g]"].to_numpy()*(2*g*radius/1000)
minus_angle = mechanic_data["Angle (-) [rad]"].to_numpy()

total_torque = np.append(minus_torque[::-1], plus_torque)
total_angle = np.append(minus_angle[::-1], plus_angle)

torque_coeff, torque_cov = curve_fit(linear, total_angle, total_torque)
torque_unc = np.sqrt(np.diag(torque_cov))
torque_residues = total_torque - linear(total_angle, *torque_coeff)
```

In [10]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(total_angle, total_torque, color = "#0C79C4")
axis[0].plot(total_angle, linear(total_angle, *torque_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Torque Mecánico [Nm]")
axis[0].grid(True)


axis[1].scatter(total_angle, torque_residues)
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Amplitud Inicial [rad]")
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].grid(True)
```



In [11]:
```
torsion_constant, torsion_constant_unc = np.abs(torque_coeff[0]), torque_unc[0]
theoric_torsion_constant = 0.058 # Nm/rad
percentage = ((theoric_torsion_constant - torsion_constant)/theoric_torsion_constant) * 100
display(Markdown(r"El valor de la constante de torsión asociada al montaje es de $\kappa_1 = {} \pm {}$ Nm/rad, la cual
```

El valor de la constante de torsión asociada al montaje es de $\kappa_1 = 12.652247683165504 \pm 0.34989441709016506$ Nm/rad, la cual difiere un -21714.220143388797% del valor teórico.

Como se logra apreciar, este valor obtenido es (DESCRIPCIÓN GENERAL).

> **Tercera Actividad: Momento Inercial.**

Otra manera de determinar la constante de torsión del experimento, así como el momento de inercia inicial del sistema (sin la inclusión de las masas), se consigue mediante la expresión:

$$\left(\frac{T}{2\pi}\right)^2 = \frac{1}{\kappa}(I_0 + n\Delta I)$$

$$\left(\frac{T}{2\pi}\right)^2 = \frac{I_0}{\kappa} + \frac{n\Delta I}{\kappa}$$

donde el ΔI representa el momento de inercia asociado a una sola de los cuadrantes:

$$\Delta I = \frac{M(R_2^2 + R_1^2)}{2}$$

y κ, la constante de torsión del sistema. El código que realiza estos cálculos es:

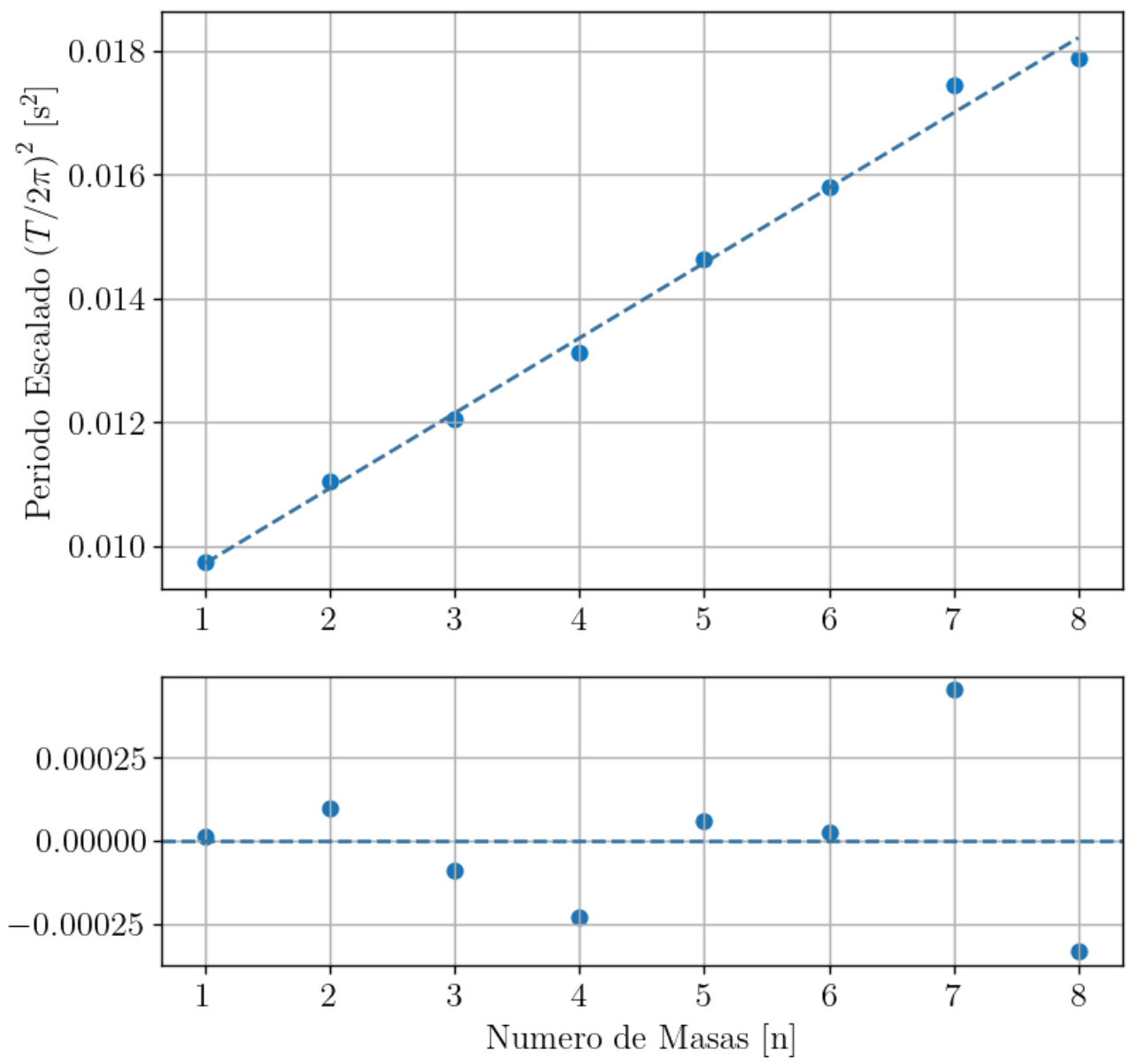
In [12]:
```
inertia = "Inertia.xlsx"
inertia_data = pd.read_excel(path + inertia)
```

Conocemos los valores de las masas, sus radios y sus incertidumbres asociadas:

In [13]:
```
M = 0.212 # Kilogramos
R1 = 0.0225 # Metros
R2 = 0.047 # Metros

dI = (np.pi**2)*(M)*((R1**2) + (R2**2))/2
```

In [14]:
```
number_masses = inertia_data["Masses [n]"].to_numpy()
period = inertia_data["Period [s]"].to_numpy()
period_scaled = (period/(4*np.pi))**2

inertia_coeff, inertia_cov = curve_fit(linear, number_masses, period_scaled)
inertia_unc = np.sqrt(np.diag(inertia_cov))
inertia_residues = period_scaled - linear(number_masses, *inertia_coeff)
```

In [15]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(number_masses, period_scaled, color = "#0C79C4")
axis[0].plot(number_masses, linear(number_masses, *inertia_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel(r"Periodo Escalado $\left(T/2\pi\right)^{2}$ [s$^{2}$]")
axis[0].grid(True)


axis[1].scatter(number_masses, inertia_residues)
axis[1].set_ylabel("Residuales")
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_xlabel("Numero de Masas [n]")
axis[1].grid(True)
```



In [16]:
```
second_torsion, second_torsion_unc = dI/np.abs(inertia_coeff[0]), inertia_unc[0]
deltaI, deltaI_unc = inertia_coeff[0], inertia_unc[0]
initial_inertia, initial_inertia_unc = inertia_coeff[1], inertia_unc[1]

inertia_percentage = ((theoric_torsion_constant - second_torsion)/theoric_torsion_constant) * 100
display(Markdown(r"El valor de la constante de torsión asociada al montaje es de $\kappa_2 = {} \pm {}$ Nm/rad (para el
```

El valor de la constante de torsión asociada al montaje es de $\kappa_2 = 2.3444108118758313 \pm 3.945404968927557e-05$ Nm/rad (para el segundo método), la cual difiere un -3942.087606682468% del valor teórico.

In [17]:
```
display(Markdown(r"Los valores encontrados para la constante de torsión son, para el primer método, $\kappa_1 = {} \pm
```

Los valores encontrados para la constante de torsión son, para el primer método,
$\kappa_1 = 12.652247683165504 \pm 0.34989441709016506$ Nm/rad y para el segundo método
$\kappa_2 = 2.3444108118758313 \pm 3.945404968927557e-05$ Nm/rad.

> **Cuarta Actividad: Torque Magnético.**
>
> En este apartado nos centramos en estudiar el comportamiento del oscilador torsional una vez las bobinas de Helmholtz se encuentran accionadas. Considerando el flujo de una corriente DC a lo largo de las bobinas, el campo magnético generado es uniforme, lo que hace que los imanes "conectados" al alambre del oscilador torsional se orienten en función de la dirección de este campo. Se llevará a cabo el siguiente procedimiento:
>
> **1)** Se registrará una muestra de mínimo 20 datos en los que hayan valores de corriente positiva y negativa. Así, se debe construir una gráfica de corriente I en función del cambio de la posición de equilibrio $\Delta\theta$.
>
> **2)** A partir de la gráfica, realizar una regresión lineal en las regiones que así lo permita, además de hablar al respecto de este comportamiento.
>
> - **Comentario:**
>
> **3)** Determinar el valor del momento magnético del imán central partiendo de la relación:
>
> $$\theta = \mu \frac{k}{\kappa} I$$
>
> donde μ es el momento magnético, $k = 3234 \times 10^{-6}$ T/A la constante de Helmholtz de las bobinas y κ la constante de torsión calculada.

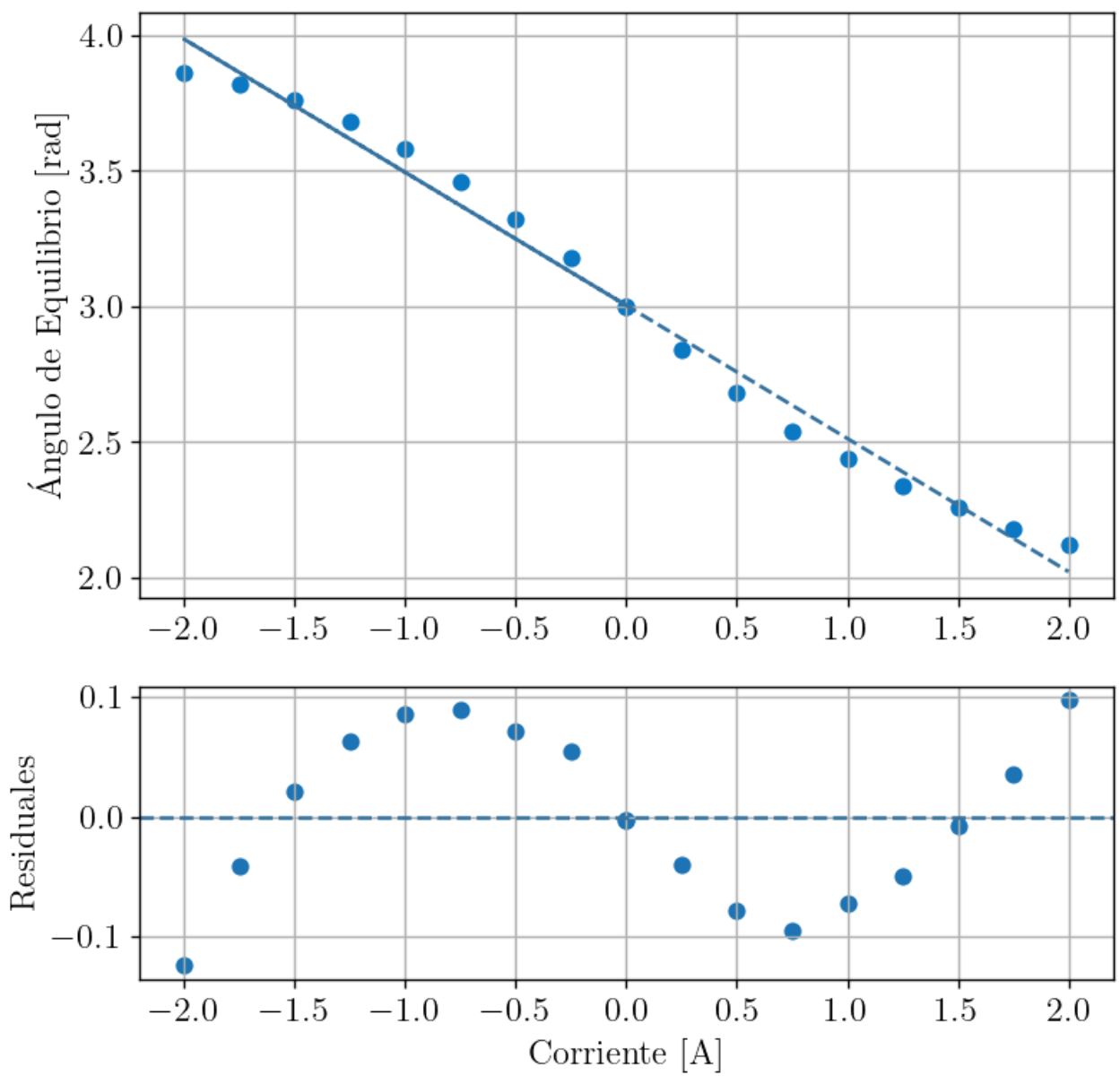
In [18]:
```
magnetic_torq = "Magnetic_Torque.xlsx"
magnetic_data = pd.read_excel(path + magnetic_torq)
magnetic_data
```

Out[18]:

	Current (+) [A]	Angle (+) [rad]	Current (-) [A]	Angle (-) [rad]
0	0.00	3.00	-2.00	3.86
1	0.25	2.84	-1.75	3.82
2	0.50	2.68	-1.50	3.76
3	0.75	2.54	-1.25	3.68
4	1.00	2.44	-1.00	3.58
5	1.25	2.34	-0.75	3.46
6	1.50	2.26	-0.50	3.32
7	1.75	2.18	-0.25	3.18
8	2.00	2.12	0.00	3.00

In [19]:
```
magnet_currentplus = magnetic_data["Current (+) [A]"]
magnet_angleplus = magnetic_data["Angle (+) [rad]"]
magnet_currentminus = magnetic_data["Current (-) [A]"]
magnet_angleminus = magnetic_data["Angle (-) [rad]"]

magnet_Tcurrent = np.append(magnet_currentminus[::-1], magnet_currentplus)
magnet_Tangle = np.append(magnet_angleminus[::-1], magnet_angleplus)

magnet_coeff, magnet_cov = curve_fit(linear, magnet_Tcurrent, magnet_Tangle)
magnet_unc = np.sqrt(np.diag(magnet_cov))
magnet_residues = magnet_Tangle - linear(magnet_Tcurrent, *magnet_coeff)
```

In [44]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(magnet_Tcurrent, magnet_Tangle, color = "#0C79C4")
axis[0].plot(magnet_Tcurrent, linear(magnet_Tcurrent, *magnet_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Ángulo de Equilibrio [rad]")
axis[0].grid(True)


axis[1].scatter(magnet_Tcurrent, magnet_residues)
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Corriente [A]")
axis[1].grid(True)
```



> **Quinta Actividad: Torque Magnético.**

En este apartado nos centramos en estudiar el comportamiento del oscilador torsional una vez se aplica un amortiguamiento mediante los imanes anexados en la zona lateral del montaje. Se busca determinar información del factor de calidad de éstas oscilaciones. La

expresión base para la obtención de las regresiones es:

$$\theta(t) = Ae^{-bt/2}\cos(ct + d)$$

donde sabemos que $A = V_0$, $b = \gamma$, $c = \omega$ y $d = \delta$ (un desfase).Recordar que el factor de calidad será:

$$Q = \frac{\sqrt{\omega^2 + (\gamma/2)^2}}{\gamma}$$

El proceso realizado es el siguiente:

In [21]:
```
def Dumping(X, a, b, c, d):
    return a*(np.e)**(b*X/2)*np.cos(c*X + d)
```

In [22]:
```
first_data = "First Measure.csv"
first_pack = pd.read_csv(path_ + first_data)
first_pack
```

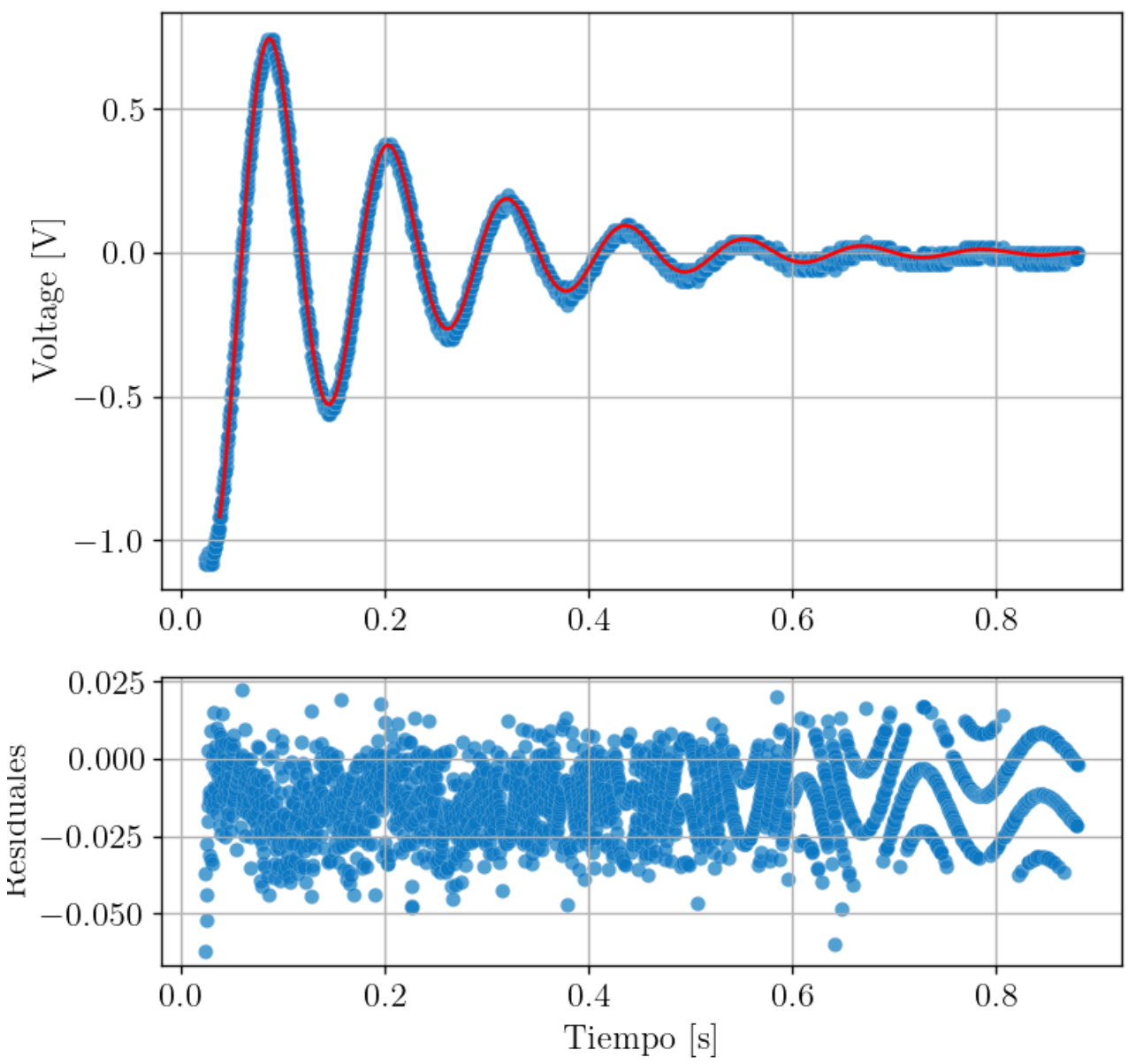
Out[22]:

	Time [s]	Voltage [V]
0	0.004000	-1.06
1	0.008000	-1.04
2	0.012000	-1.06
3	0.016000	-1.08
4	0.020000	-1.06
...	...	...
2494	9.980000	-2.56
2495	9.984000	-1.06
2496	9.988001	-1.06
2497	9.992001	-1.06
2498	9.996000	-1.08

2499 rows × 2 columns

In [23]:
```
bins = 30
ficsit = 30
fTime = first_pack["Time [s]"].to_numpy()[bins:-300]/10
_fTime_ = np.linspace(fTime[0], fTime[-1], 1000)
fVoltage = first_pack["Voltage [V]"].to_numpy()[bins:-300]

fTime_coeff, fTime_cov = curve_fit(Dumping, fTime[ficsit:], fVoltage[ficsit:])
fTime_unc = np.sqrt(np.diag(fTime_cov))
fTime_residues = fVoltage - Dumping(fTime, *fTime_coeff)
```

In [24]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(fTime[ficsit:], fVoltage[ficsit:], color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0
axis[0].plot(_fTime_[ficsit:], Dumping(_fTime_, *fTime_coeff)[ficsit:], color = "red")
axis[0].set_ylabel(r"Voltage [V]")
axis[0].grid(True)

axis[1].grid(True)
axis[1].scatter(fTime[ficsit:], fTime_residues[ficsit:], color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alp
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Tiempo [s]")
```



Out[24]:
```
Text(0.5, 0, 'Tiempo [s]')
```

In [25]:
```
Q = np.sqrt((fTime_coeff[2]**2) + (fTime_coeff[1]/2)**2)/np.abs(fTime_coeff[1])
display(Markdown(r"El factor de calidad de esta oscilación es de {}.".format(Q)))
```

El factor de calidad de esta oscilación es de 4.590306459857158.

In [26]:
```
second_data = "Second Measure.csv"
second_pack = pd.read_csv(path_ + second_data)
second_pack
```

Out[26]:

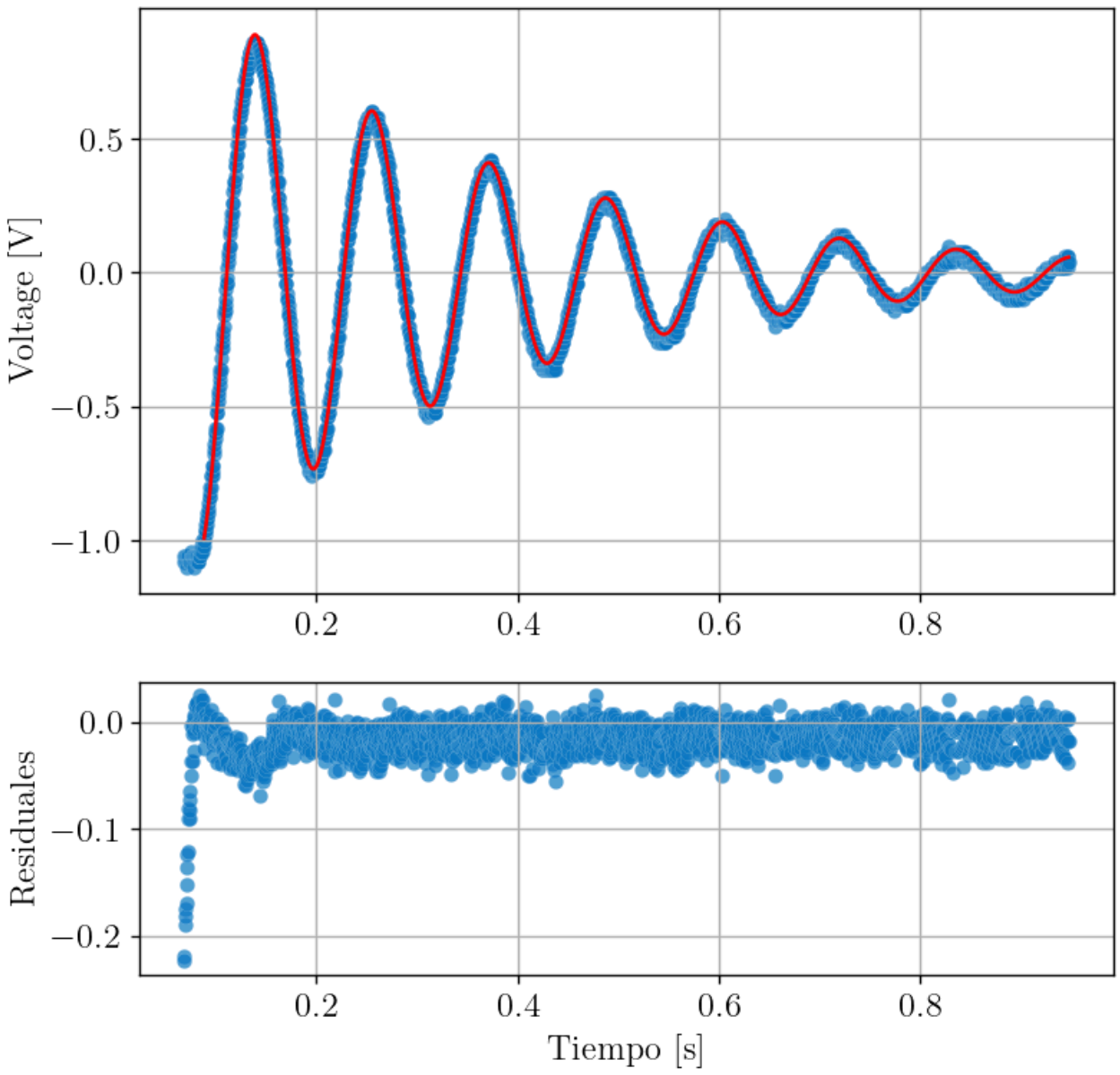
	Time [s]	Voltage [V]
0	0.004000	-2.56
1	0.008000	-2.56
2	0.012000	-2.56
3	0.016000	-2.56
4	0.020000	-2.56
...	...	...
2494	9.980000	-2.56
2495	9.984000	-2.56
2496	9.988001	-2.56
2497	9.992001	-2.56
2498	9.996000	-2.56

2499 rows × 2 columns

In [27]:
```
sbins = 130
sficsit = 40
sTime = second_pack["Time [s]"].to_numpy()[sbins:-sbins]/10
_sTime_ = np.linspace(sTime[0], sTime[-1], 1000)
sVoltage = second_pack["Voltage [V]"].to_numpy()[sbins:-sbins]

sTime_coeff, sTime_cov = curve_fit(Dumping, sTime[sficsit:], sVoltage[sficsit:])
sTime_unc = np.sqrt(np.diag(sTime_cov))
sTime_residues = sVoltage - Dumping(sTime, *sTime_coeff)
```

In [28]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(sTime[sficsit:], sVoltage[sficsit:], color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha =
axis[0].plot(_sTime_[sficsit:], Dumping(_sTime_, *sTime_coeff)[sficsit:], color = "red")
axis[0].set_ylabel(r"Voltage [V]")
axis[0].grid(True)
```

```
axis[1].grid(True)
axis[1].scatter(sTime[sficsit:], sTime_residues[sficsit:], color = "#0C79C4", edgecolors = "white", linewidths = 0.1, a
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Tiempo [s]")
```



Out[28]:
```
Text(0.5, 0, 'Tiempo [s]')
```

In [29]:
```
Q = np.sqrt((sTime_coeff[2]**2) + (sTime_coeff[1]/2)**2)/np.abs(sTime_coeff[1])
display(Markdown(r"El factor de calidad de esta oscilación es de {}.".format(Q)))
```

El factor de calidad de esta oscilación es de 8.146092128917747.

In [30]:
```
third_data = "Third Measure.csv"
third_pack = pd.read_csv(path_ + third_data)
third_pack
```

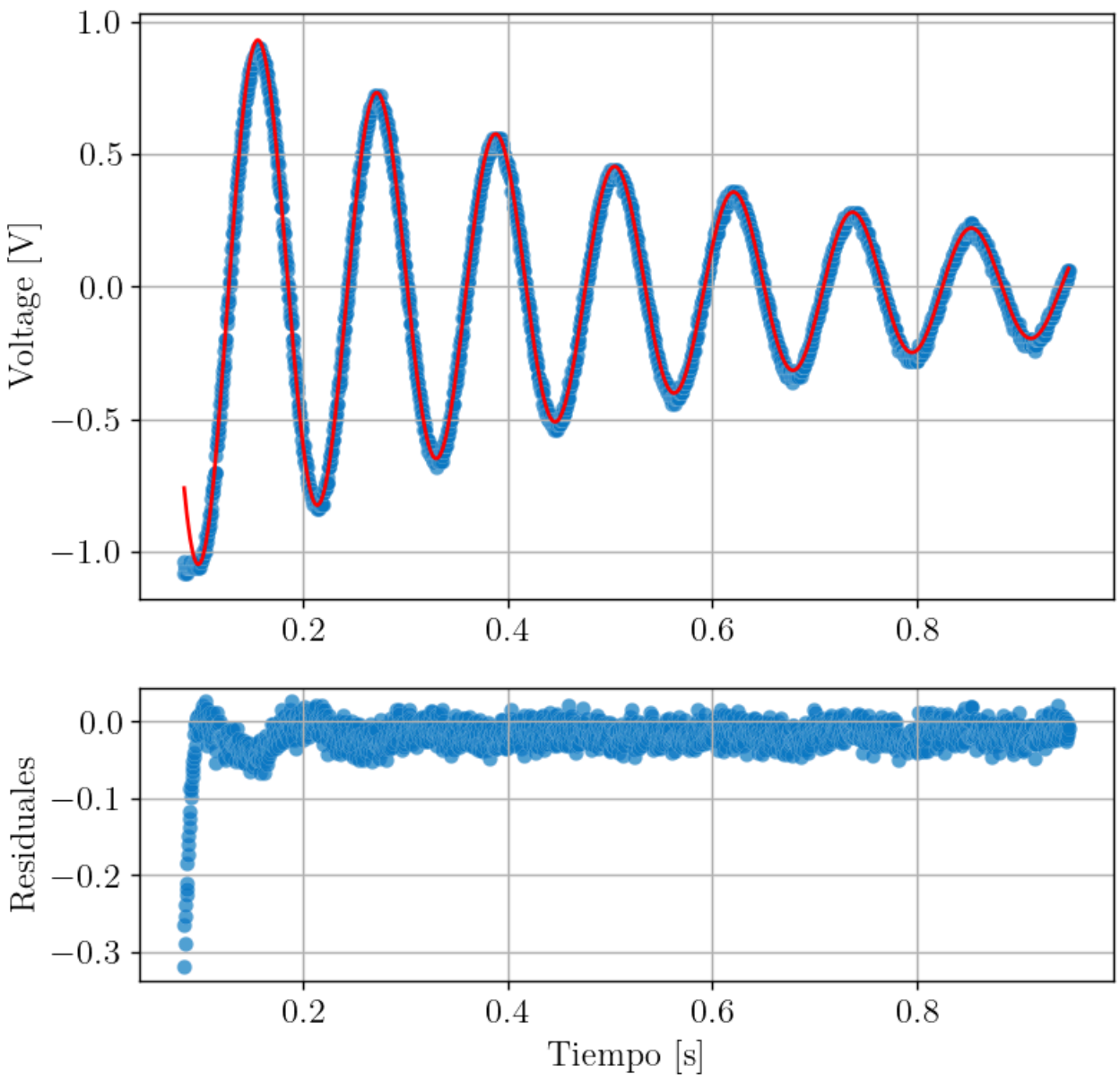
Out[30]:

	Time [s]	Voltage [V]
0	0.004000	-2.56
1	0.008000	-2.56
2	0.012000	-2.56
3	0.016000	-2.56
4	0.020000	-2.56
...	...	...
2494	9.980000	-2.56
2495	9.984000	-2.56
2496	9.988001	-2.56
2497	9.992001	-2.56
2498	9.996000	-2.56

2499 rows × 2 columns

In [31]:
```
tbins = 130
tTime = third_pack["Time [s]"].to_numpy()[tbins + 80:-tbins]/10
_tTime_ = np.linspace(tTime[0], tTime[-1], 1000)
tVoltage = third_pack["Voltage [V]"].to_numpy()[tbins + 80:-tbins]

tTime_coeff, tTime_cov = curve_fit(Dumping, tTime, tVoltage)
tTime_unc = np.sqrt(np.diag(tTime_cov))
tTime_residues = tVoltage - Dumping(tTime, *tTime_coeff)
```

In [32]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(tTime, tVoltage, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7)
axis[0].plot(_tTime_, Dumping(_tTime_, *tTime_coeff), color = "red")
axis[0].set_ylabel(r"Voltage [V]")
axis[0].grid(True)

axis[1].grid(True)
axis[1].scatter(tTime, tTime_residues, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7)
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Tiempo [s]")
```



Out[32]:
```
Text(0.5, 0, 'Tiempo [s]')
```

In [33]:
```
Q = np.sqrt((tTime_coeff[2]**2) + (tTime_coeff[1]/2)**2)/np.abs(tTime_coeff[1])
display(Markdown(r"El factor de calidad de esta oscilación es de {}.".format(Q)))
```

El factor de calidad de esta oscilación es de 13.116761177765735.

> **Sexta Actividad: Resonancia.**

Habiendo realizado un barrido de frecuencias en las zonas de 700 mHz a 1000 mHz, para tres escenarios con amortiguamiento. La expresión del ajuste tendrá la forma:

$$A = \frac{A_0}{\sqrt{(\omega_0^2 - \omega^2)^2 + (\gamma\omega)^2}}$$

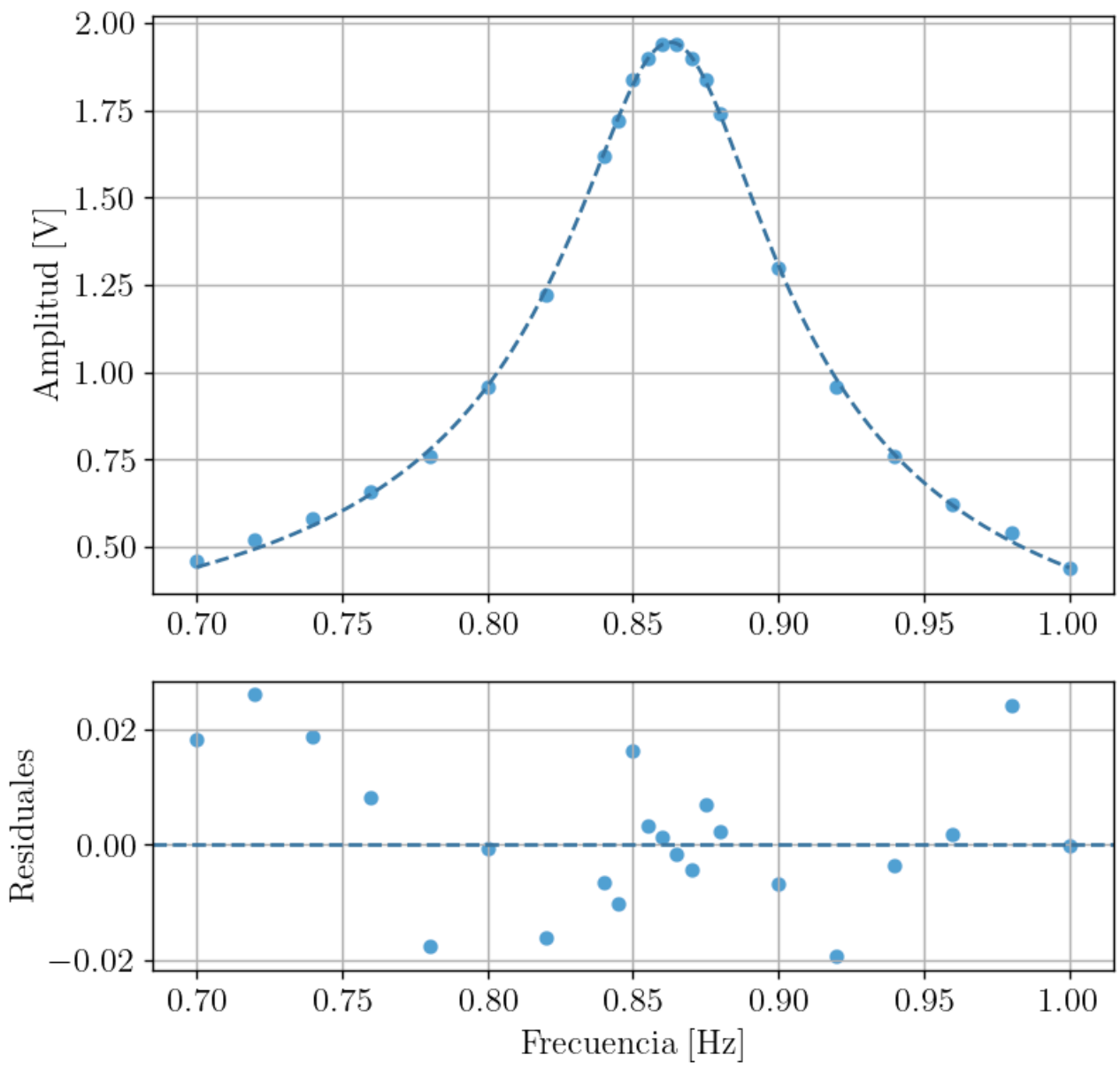
In [77]:
```
def Amplitude(X, a, b, c):
    return a/np.sqrt(((b**2) - (X**2))**2 + (c*X)**2)
```

In [93]:
```
resonance = "Resonance.xlsx"
resonance_data = pd.read_excel(path + resonance)
```

In [87]:
```
ffrequency = resonance_data["1. Frequency [mHz]"].to_numpy()/1000
_ffrequency_ = np.linspace(ffrequency[0], ffrequency[-1], 1000)
ffvoltage = resonance_data["1. Vpp [mV]"].to_numpy()/1000

ffrequency_coeff, ffrequency_cov = curve_fit(Amplitude, ffrequency, ffvoltage)
ffrequency_unc = np.sqrt(np.diag(ffrequency_cov))
ffrequency_residues = ffvoltage - Amplitude(ffrequency, *ffrequency_coeff)
```

In [88]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(ffrequency,ffvoltage, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7)
axis[0].plot(_ffrequency_, Amplitude(_ffrequency_, *ffrequency_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Amplitud [V]")
axis[0].grid(True)
```

```
axis[1].grid(True)
axis[1].scatter(ffrequency, ffrequency_residues, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Frecuencia [Hz]")
```



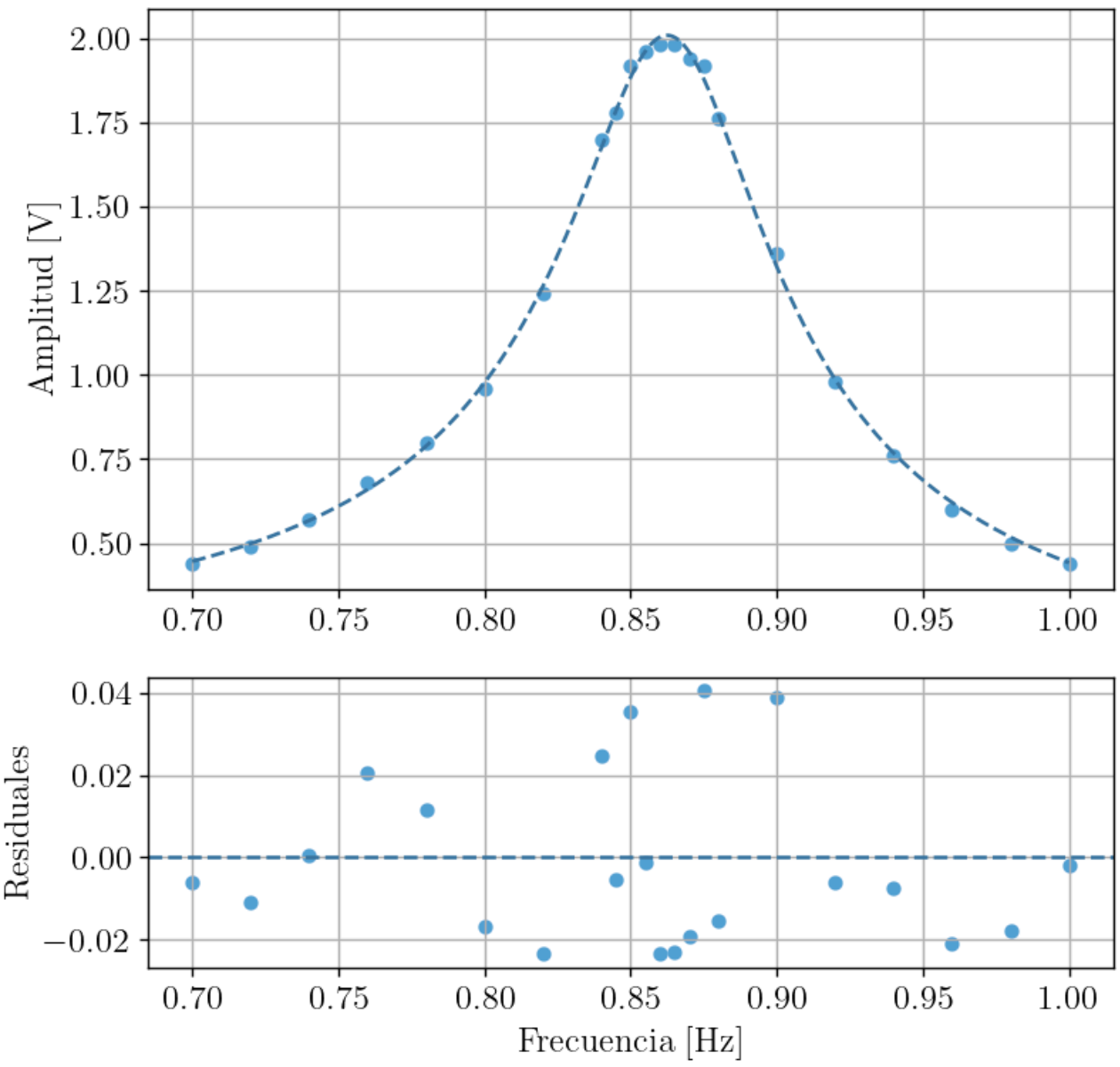
Out[88]: `Text(0.5, 0, 'Frecuencia [Hz]')`

In [99]:
```
f_FResonance = ffrequency_coeff[1], ffrequency_unc[1]
display(Markdown(r"La frecuencia de resonancia de éste sistema {} +/- {} mHz.".format(f_FResonance[0]*1000, f_FResonanc
```

La frecuencia de resonancia de éste sistema 864.2312310994137 +/- 0.23532484216183638 mHz.

In [89]:
```
sfrequency = resonance_data["2. Frequency [mHz]"].to_numpy()/1000
_sfrequency_ = np.linspace(sfrequency[0], sfrequency[-1], 1000)
ssvoltage = resonance_data["2. Vpp [mV]"].to_numpy()/1000

sfrequency_coeff, sfrequency_cov = curve_fit(Amplitude, sfrequency, ssvoltage)
sfrequency_unc = np.sqrt(np.diag(sfrequency_cov))
sfrequency_residues = ssvoltage - Amplitude(sfrequency, *sfrequency_coeff)
```

In [90]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(sfrequency, ssvoltage, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7)
axis[0].plot(_sfrequency_, Amplitude(_sfrequency_, *sfrequency_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Amplitud [V]")
axis[0].grid(True)

axis[1].grid(True)
axis[1].scatter(sfrequency, sfrequency_residues, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Frecuencia [Hz]")
```



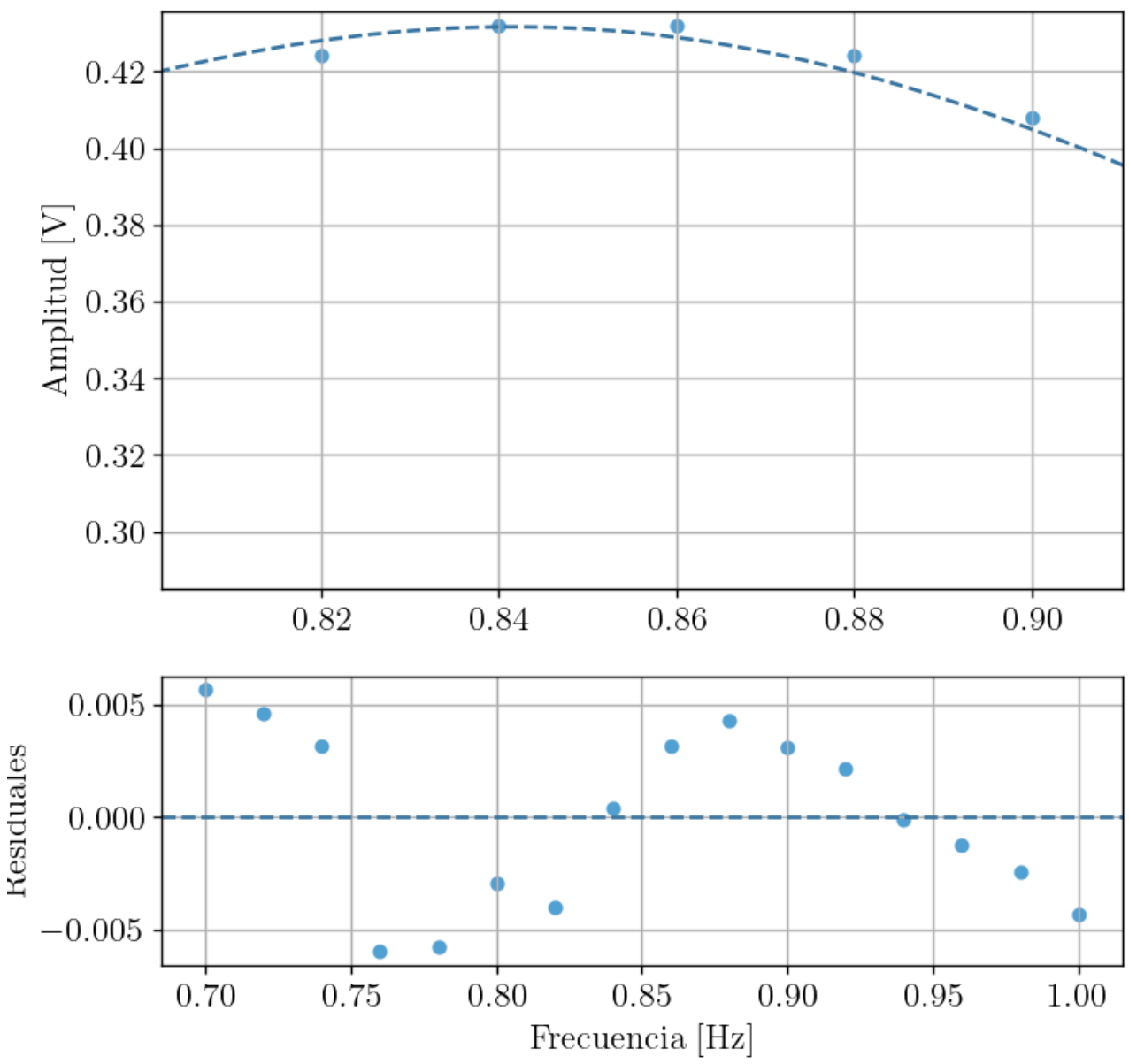
Out[90]:
```
Text(0.5, 0, 'Frecuencia [Hz]')
```

In [100…
```
s_FResonance = sfrequency_coeff[1], sfrequency_unc[1]
display(Markdown(r"La frecuencia de resonancia de éste sistema {} +/- {} mHz.".format(s_FResonance[0]*1000, s_FResonanc
```

La frecuencia de resonancia de éste sistema 863.7479166798078 +/- 0.3578219299240177 mHz.

In [91]:
```
tfrequency = resonance_data["3. Frequency [mHz]"].to_numpy()[:-6]/1000
_tfrequency_ = np.linspace(tfrequency[0], tfrequency[-1], 1000)
ttvoltage = resonance_data["3. Vpp [mV]"].to_numpy()[:-6]/1000

tfrequency_coeff, tfrequency_cov = curve_fit(Amplitude, tfrequency, ttvoltage)
tfrequency_unc = np.sqrt(np.diag(tfrequency_cov))
tfrequency_residues = ttvoltage - Amplitude(tfrequency, *tfrequency_coeff)
```

In [92]:
```
figure, axis = plt.subplots(2, 1, figsize = (7, 7), gridspec_kw={'height_ratios': [2, 1]})
axis[0].scatter(tfrequency, ttvoltage, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7)
axis[0].plot(_tfrequency_, Amplitude(_tfrequency_, *tfrequency_coeff), color = "#3875A0", linestyle = "--")
axis[0].set_ylabel("Amplitud [V]")
axis[0].grid(True)

axis[1].grid(True)
axis[1].scatter(tfrequency, tfrequency_residues, color = "#0C79C4", edgecolors = "white", linewidths = 0.1, alpha = 0.7
axis[1].axhline(0, color = "#3875A0", linestyle = "--")
axis[1].set_ylabel("Residuales")
axis[1].set_xlabel("Frecuencia [Hz]")
```



Out[92]: Text(0.5, 0, 'Frecuencia [Hz]')

In [101…
```
t_FResonance = tfrequency_coeff[1], tfrequency_unc[1]
display(Markdown(r"La frecuencia de resonancia de éste sistema {} +/- {} mHz.".format(t_FResonance[0]*1000, t_FResonanc
```

La frecuencia de resonancia de éste sistema 871.7538134155533 +/- 1.4048366554505751 mHz.